Bachelor thesis
Computer Science

Radboud University

Classifying Landsat Terrain Images via Random Forests

Author:
Koen Basten
4119657
thesis@kbasten.com

First supervisor/assessor:
Fabian Gieseke
fgieseke@cs.ru.nl

Second assessor:
Arjen de Vries
arjen@acm.org

August 15, 2016

Abstract

In this thesis we propose a method for classifying various terrain types from Landsat imagery using random forest classification. OpenStreetMap is used to generate training data for the classifier, as well as to define areas for extracting statistics about the underlying terrain. This automated approach can be applied to large sets of data, reducing the need for manual labeling.

Contents

1 Introduction

The Landsat program run by NASA since 1972 provides satellite imagery to researchers. The program aims for complete world coverage. Of the eight satellites in this program, the two most recent satellites are still active[1] and achieve complete coverage every eight days. The data sets produced by these satellites can be downloaded using an online portal. The program lists several areas of research their imagery is used for, including land use and mapping, environmental monitoring, geology, and coastal resources [8].

The latest satellite in the program, Landsat 8, provides 11 different multispectral bands at a resolution of 30 by 30 meters. This spatial resolution is far from the best that is (commercially) available. However, the spectral resolution and amount of coverage it provides is not found in any other satellite imagery aqcuisition program. This makes Landsat a great source of information for comprehensive studies about the geography of the earth and its terrain. The longevity of the Landsat program also enables researchers to conduct studies over long periods of time. A new satellite for the program is scheduled to launch in 2023.

In this thesis we will focus on creating land cover maps using random forests. Land cover maps visually show how an area is covered by certain features like forests, grassland, or buildings. These maps are utilised by various studies such as assessing ecosystem status and health, and monitoring urban development [20]. Automated methods for generating land cover maps are required for these purposes, as for areas with rapidly changing terrain conditions a manual approach would be very labor intensive. Several specialised applications have been developed for classifying very detailed land cover maps [19], however the satellite imagery used does not provide complete world coverage and thus can not be applied to any location on the globe.

With land cover maps, statistics about the underlying terrain can be obtained. These statistics can then be used to monitor urban growth or document ecological conditions [15] for the length of the Landsat program. While the focus of this thesis is on land cover map generation, we will also explore how we can extract statistics for the resulting maps.

[1] Older satellites either failed or were decomissioned.

2 Preliminaries

In this chapter we will discuss the knowledge needed to read this paper. Some knowledge on machine learning and classification algorithms is assumed. However an explanation of decision trees and random forest classification follows.

2.1 OpenStreetMap

OpenStreetMap is a crowdsourced web application for creating a map of the world. Their maps are accessible and editable by anybody for any purpose. The map contains several hundred different features, a subset of which we will use to train a classifier. The degree of coverage and accuracy of OpenStreetMap varies greatly per country. Not only are physical features mapped, but also administrative borders, mapping city, state, and country borders. We will make extensive use of this project and its public API to generate maps.

2.2 Decision tree learning

Decision tree learning aims to create a model of decisions that map input to labels. It is represented much like a directed acyclic graph. Figure 2.1 is an example of a decision tree. To obtain a class for any input, the tree is traversed top to bottom. At every node, the criterion at that node is checked against the value of the input. From there, the input is either redirected to the left branch, or to the right branch, based on the condition at that node. Whenever the bottom of the tree is reached in this manner, a class is returned for the input. Decision trees can be automatically generated from labeled data sets. The Python package `scikit-learn` provides excellent decision tree classifier implementations.

2.3 Random forest classification

Random forests are a collection of decision trees aimed to be an improvement in performance over single decision trees. A random forest is constructed by generating decision trees for subsamples of the data. Classifying is then performed much in the same way as with regular decision trees, where the resulting class is the mean class of all the single decision trees in the forest. A random forest can contain an arbitrary number of trees. We will explore how the quality of the resulting classification changes with a varying number of trees in the forest.

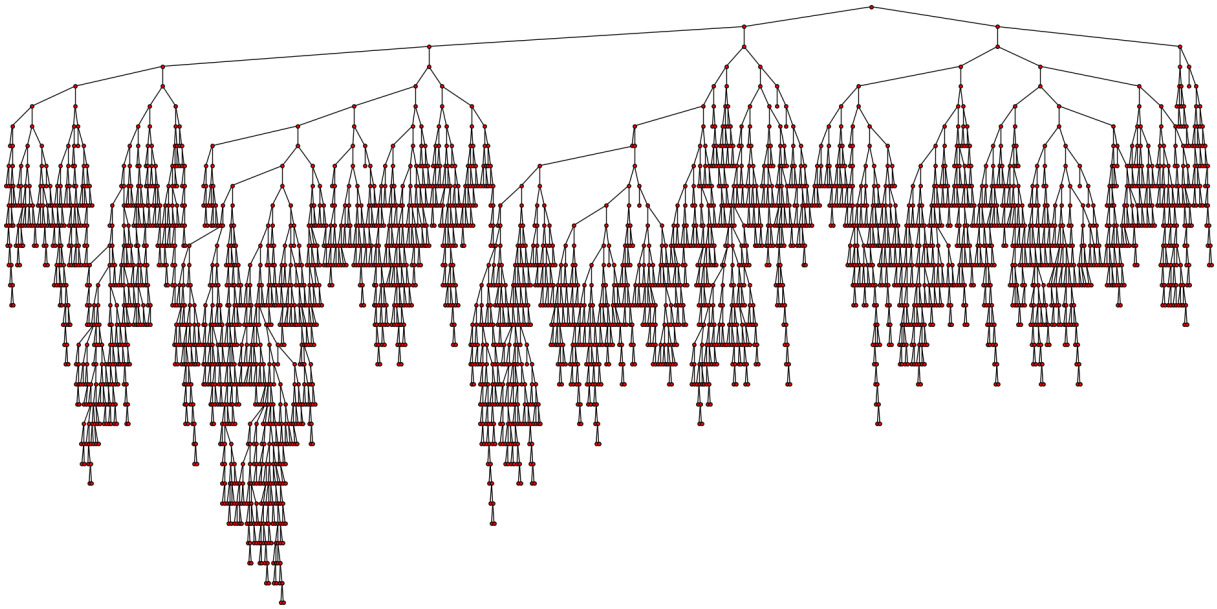


Figure 2.1: Decision tree

The trees in a random forest are generated in a way that attempts to split the data set at every node in half. The algorithm selects at every node the feature, and a value for that feature, that best accomplishes this goal from a set of $n = \sqrt{n\_features}$ features. We will go into further detail on random forest generation in Figure 4.3.1.

3 Data

3.1 Projections

To display coordinates from a sphere on flat surfaces like computer screens, a transformation of coordinates is required. An example of this tranformation is the Mercator projection, which projects the surface of the earth on a cylinder, which can be rolled out to a flat surface. This projection is not ideal for satellite images, since it disproportionally stretches the top and bottom of the sphere. In the case of the Mercator projection, this means the north- and south pole are stretched along the entire length of the projection. A consequence of this is that measuring distances between two points on this map is inaccurate.

The EPSG Geodetic Parameter Registry hosts a number of coordinate transformation systems for online reference [5]. The Python package `pyproj` lets us transform any latitude-longitude coordinate to an x-y coordinate suited for displaying on screen, given an EPSG code. We will make extensive use of this package to transform coordinates between various systems for the purpose of generating training data.

3.1.1 Landsat 8

All Landsat imagery uses a subset of the *WGS 84* EPSG standard. Metadata provided with the Landat images contains the exact EPSG code which was used to transform the original image. We can extract this code from the metadata to later apply the transformation of this code to OpenStreetMap coordinates for the purpose of lining up OpenStreetMap coordinates and Landsat images. In Figure 3.1 we can see the river Thames in blue from the OpenStreetMap data, plotted over a satellite image of that area. In Section 4.2 we will go into detail on this process.

WGS 84 is the primary standard used by the Global Positioning System (GPS). Developed by the U.S. Department of Defense, it was later implemented by Landsat for its coordinate system. This system is used for every Landsat image, regardless of its position on the globe.

3.1.2 OpenStreetMap

The OpenStreetMap website uses the EPSG code *3857*, which is a subset of the WGS 84 standard[1]. This system is used to transform between latitude-longitude and x-y coordinate systems.

[1] EPSG 3857 is also known as Web Mercator and heavily used in web applications like Google Maps and OpenStreetMap.

Figure 3.1: Satellite image of London with OpenStreetMap waterways

We can use the EPSG code provided by the Landsat image metadata to transform the OpenStreetMap coordinates to the same system as the image in a similar fashion as described above using `pyproj`, resulting in Figure 3.1. Transformations like this are required for mapping OpenStreetMap labels to their actual positions on a satellite image which we can then use for training a classifier.

3.2 OpenStreetMap

The OpenStreetMap archive provides downloads for their entire data set[2]. The data is a collection of the following components, taken from the OpenStreetMap wiki [16]:

- nodes (defining points in space),
- ways (defining linear features and area boundaries), and
- relations (which are used to explain how other elements work together).

The *relations* component is most commonly used to define relationships between *ways*. In OpenStreetMap, administrative boundaries mark country borders as well as region, district/county, and city borders and are defined as *relations* between *ways*. We will use these relations for deriving statistics from classified images as detailed in Section 4.5. Relations are not related to physical objects and thus can not be detected on satellite images. Because of this, when parsing OpenStreetMap data for training our model, we can skip relations and only use nodes and ways.

As nodes define single points in space, they are used to represent lamp posts, park benches, fire hydrants, and other small elements alike. Since the spatial resolution of the Landsat satellites is not sufficient for classifying such small objects, we can safely ignore them. Hence, the node elements are only used in conjunction with a *way* representing an area[3] or line[4]. Nodes not linked to a way are ignored. This leaves only *ways* to be parsed.

3.2.1 Training map generation

We use Python to generate an image which is used to map OpenStreetMap labels to corresponding pixels on a satellite image. To do so, we need to generate an image which is overlaid on top of the satellite image, where every pixel of our image relates to a single pixel of the satellite image. The OpenStreetMap data can be parsed using the Python library `imposm.parser`[5] and drawn to an image. Figure 3.2 shows a subset of the OpenStreetMap data for the London area with roads in black, railroads in red and water in blue. This map is drawn using the EPSG code for this area, taken from the Landsat metadata. Using any other EPSG code will result in the map not perfectly aligning with its related satellite image. It is also possible to manually draw training maps. We will go into detail on this process in Section 4.2.1 and Section 4.2.2.

3.3 Landsat 8

The Landsat archive provides satellite imagery in 11 multispectral bands at a resolution of 30 by 30 meters. Figure 3.3 is an example of a Landsat image composited of bands 2, 3 and 4. Listed in Table 3.1 are the band designations, listing per band the wavelength range it spans [9]. Note that both thermal infrared bands are acquired at a resolution of 100 meters, but are resampled up to 30 meters to match the resolution of the other bands. Also note that the resolution of the

[2] Subsets can also be downloaded.
[3] Buildings are represented by a closed polygon.
[4] Roads are represented as a list of nodes.
[5] Documentation at `http://imposm.org/docs/imposm.parser/latest/`

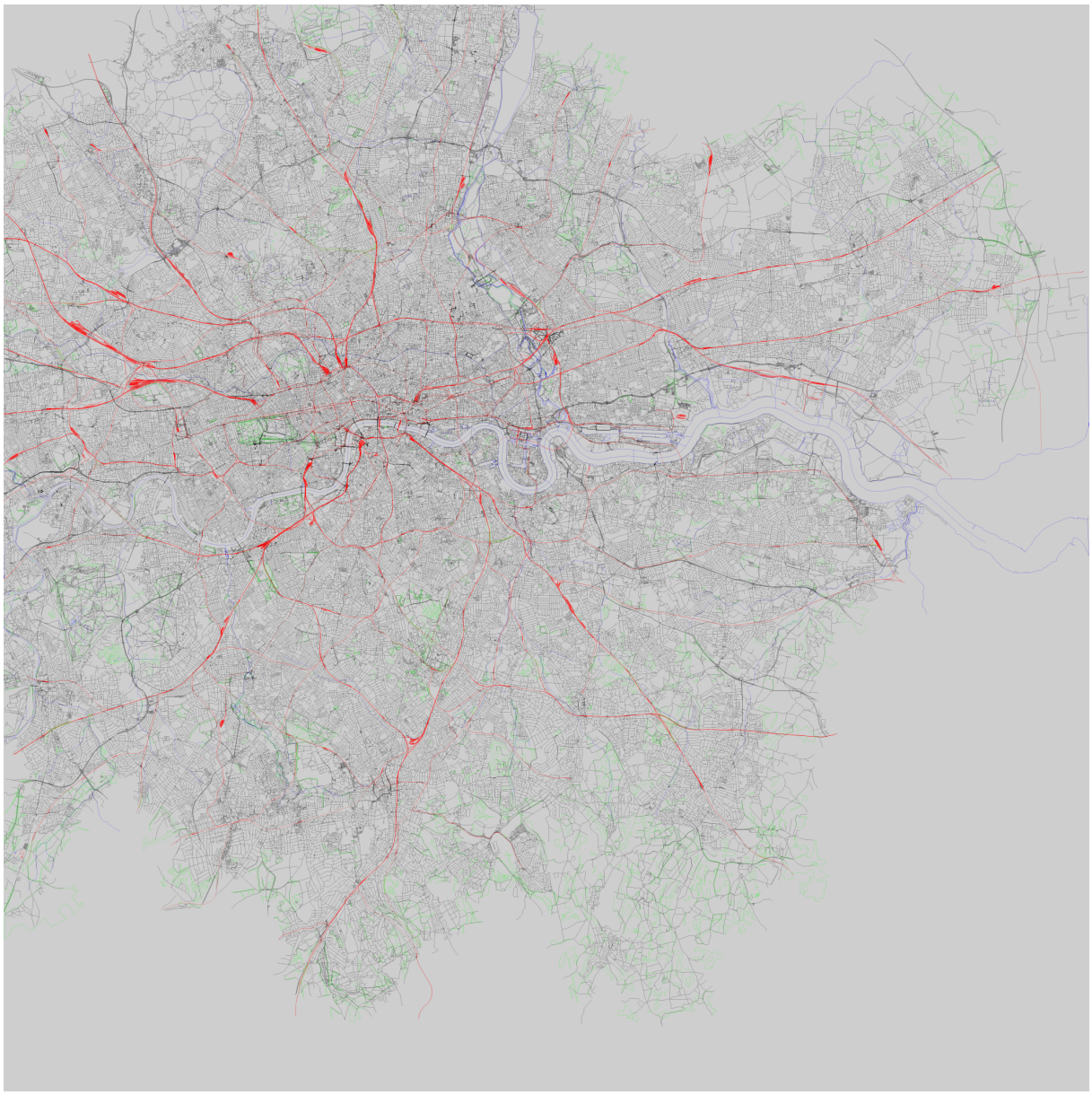


Figure 3.2: Generated image of London from OpenStreetMap data

panchromatic band is twice that of the other bands. We will later see that this band can be used to improve the resolution of the other bands through a process called pan sharpening discussed in Section 4.3.1.

The U.S. Geology Survey (USGS) department, which hosts a number of online Landsat tools, lists the type of terrain detection each band is most commonly used for. For example, band 5 *emphasizes biomass content and shorelines* [13]. However, since we let the classifier decide which features are most important in certain scenarios, we will not be implementing any band preference based on hints from the U.S. Geology Survey department. This ensures that, independent of which Landsat release is used, our method utilizes every available band in a manner that results in the best classification.

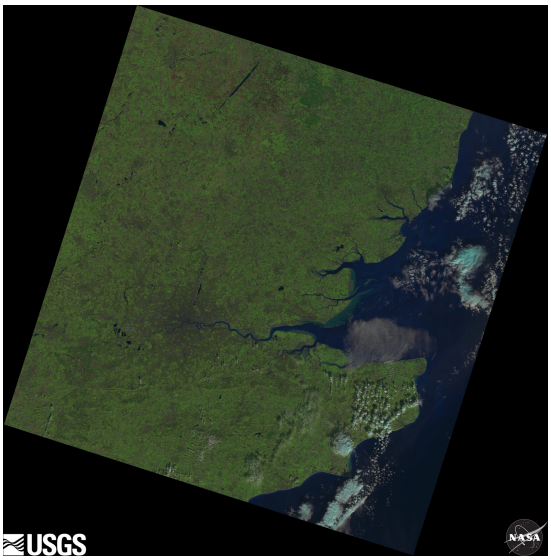


Figure 3.3: Landsat image of London

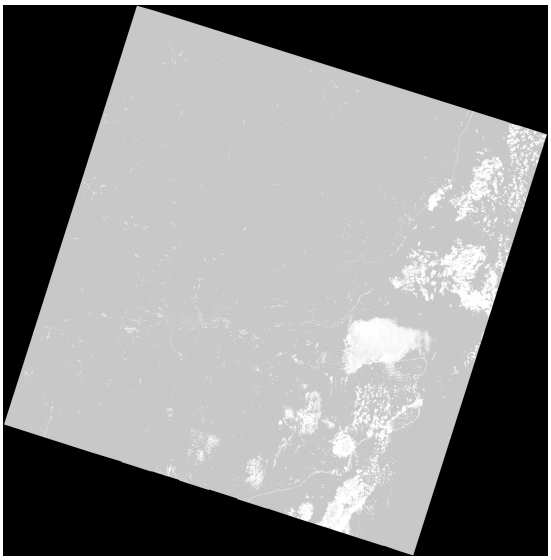


Figure 3.4: Fill and cloud mask

Bands	Wavelength (μm)	Resolution (meters)
Band 1 - Coastal aerosol	0.43 - 0.45	30
Band 2 - Blue	0.45 - 0.51	30
Band 3 - Green	0.53 - 0.59	30
Band 4 - Red	0.64 - 0.67	30
Band 5 - Near Infrared	0.85 - 0.88	30
Band 6 - SWIR 1	1.57 - 1.65	30
Band 7 - SWIR 2	2.11 - 2.29	30
Band 8 - Panchromatic	0.50 - 0.68	15
Band 9 - Cirrus	1.36 - 1.38	30
Band 10 - Thermal Infrared 1	10.60 - 11.19	30
Band 11 - Thermal Infrared 2	11.50 - 12.51	30

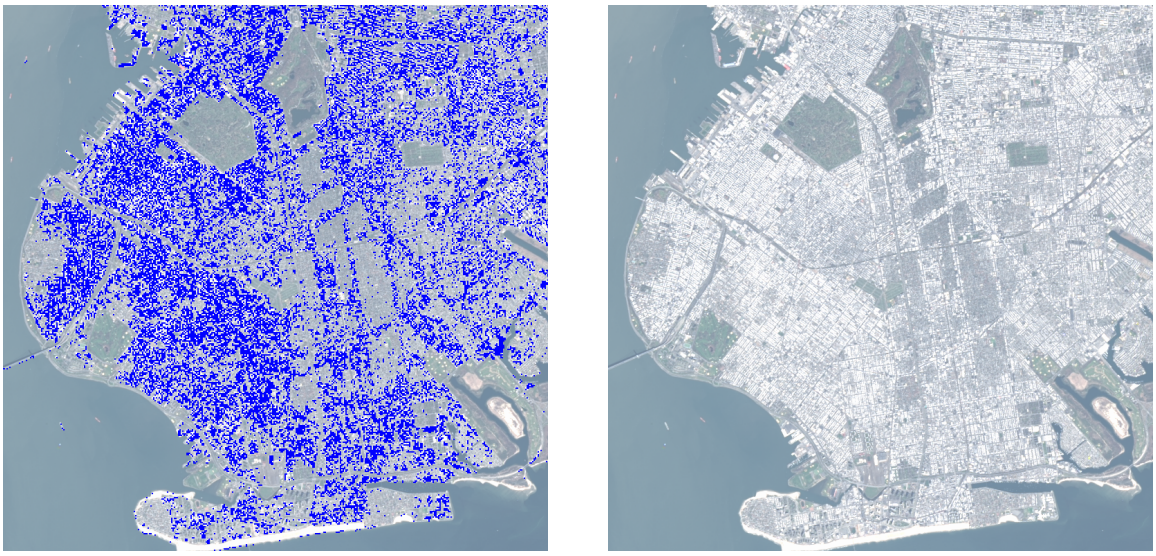
Table 3.1: Landsat 8 band designations

Landsat 8 improves on previous Landsat satellites by providing additional bands across a broader wavelength spectrum[6]. Our method works for any Landsat release as any subset of bands can be used as input for the classifier. Some problems surface when anything other than Landsat 8 is used due to availability of training data. In Figure 4.3.2 we list possible solutions.

3.3.1 Quality Assessment Band

The Landsat imagery includes a quality assessment (QA) band [11]. From this band we can derive whether a pixel is cloud-covered or not, whether the pixel is filler space or not, and a few additional sensor conditions. We will use this band to discard any pixels that do not represent

[6]See Table A.1 for a list of Landsat 7 bands



(a) Cloud mask in blue

(b) Actual scene

Figure 3.5: Built-in cloud detection errors over Brooklyn, caused by bright building tops

terrain data. For the satellite image in Figure 3.3, we see the fill mask in black and the cloud mask in white in Figure 3.4. This QA band also includes a snow mask, and water mask. For the reason listed previously, we will not implement any of the terrain hints from the QA band. Moreover, the USGS states that none of the masks are expected to exceed an accuracy of 80%[7]. Especially in urban areas close to bodies of water, we found that the built-in cloud detection errs often. See Figure 3.5 for an example with the clouds from the QA band in blue. Notice how most of the area covered by buildings is also covered by clouds, whereas any water or parks are not. No clouds are actually present in this image. This is a known problem also present for the snow/ice mask of the QA band. The Landsat program aims to have fixed this problem by summer 2016 using post-processing [10]. This fix will also introduce a mask for cloud shadows which is not present in the current QA band file.

[7]Fill mask excluded.

4 | Approach

4.1 Scenarios

We define several groupings of classes as *scenarios*. Scenarios are used to list class labels to classify in an image. Using scenarios we can generate training data, or define boundaries for later statistics calculations.

We select several of the most often used OpenStreetMap labels to define a scenario in order to test the performance our model, as discussed in Section 4.3.3. The features most commonly defined in OpenStreetMap used in this scenario include:

- roads, including all motorways, roads, and otherwise paved surfaces,
- waterways, including rivers, riverbanks, seas, and water reservoirs,
- forests,
- buildings,
- grass, including parks, and meadows,
- farmland, including crops, and vegetables.

In this thesis we describe several other scenarios such as land-water separation, and cloud detection. Any combination of features existent in OpenStreetMap[1] can be defined as scenario to suit various applications.

4.1.1 Cloud detection

While the QA band provides built-in cloud detection, we propose a scenario to label clouds in satellite images, independent of the quality assessment band cloud labels. The goal of this scenario is to try and improve on the built-in cloud detection algorithms of Landsat 8, since NASA does not recommend the built-in cloud detection be used for more than an indication of where clouds might be present. This is where an improved cloud detection method would improve the overall results of classification. Our method does not detect shadows cast by clouds, so it should be noted that without sufficient training data pixels covered by cloud shadows are likely to be misclassified due to the darker tones of the pixels. An example of the built-in cloud detection can be seen in Figure 3.4.

[1] See `http://wiki.openstreetmap.org/wiki/Map_Features` for a comprehensive list of OpenStreetMap features.

4.2 Training data

Training data for our method consists of an image with colored pixels representing classes. There are several ways to generate training maps.

The Landsat metadata provides information about image dimensions. We adhere to those dimensions when generating a training image so that we can overlap the training map and the Landsat imagery as shown in Figure 3.1. This approach enables manual verification of the training data.

4.2.1 OpenStreetMap

The OpenStreetMap data dumps contain coordinates in a geographic coordinate system. To transform those coordinates for plotting an image, we can make use of the same transformation used by Landsat to display part of a sphere on a flat surface.

We generate a training image where different colors correspond to the different labels. Colors for labels can be specified in the scenario definition file. The Landsat metadata provides us with image dimensions as well as designated fill pixels. We can make use of the cloud detection scenario to overlay cloud cover, or use the built-in cloud detection algorithm.

This method of generating training data for our classifier is highly dependent on correct data from OpenStreetMap. Currently, we have no way of verifying the accuracy of the crowdsourced data. Figure 4.1 is an example of incorrectly labeled roads. Also shown in this figure is the incompleteness of the OpenStreetMap dataset, which should not matter for our classifier as long as enough instances of every class are present.

4.2.2 Manual

The training data for our method consists of a generated image where classes are labeled using different colors. In contrast to directly using OpenStreetMap dumps as input for our classifier, this approach enables us to manually draw training images to then be parsed as training data. For several scenarios, this is the only approach that allows us to label certain classes. The cloud detection scenario is a prime example of this, since OpenStreetMap does not map any clouds. An additional benefit of this flexible approach is that it somewhat reduces the runtime of our method since the training images can be re-used across classifications of the same scenario because they are generated using a separate application. Parsing the OpenStreetMap data is only done once.

These manually defined training maps can be generated by drawing colored pixels on a satellite image where pixels line up with classes using any image editor[2]. See Figure 4.2a for an example where the green pixels cover only land, and blue pixels cover only water[3]. A classification using this scenario applied to satellite Figure 3.3 results in Figure 4.2b. This approach is not practical for classifying small objects like highways since labeling training instances for these classes pixel by pixel is very time consuming. Manually drawing/augmenting training maps from Section 4.2.1 allows fixing incorrect and incomplete OpenstreetMap data.

[2]Image dimensions should equal satellite image dimensions.
[3]The Landsat provided fill mask and cloud map is also shown.

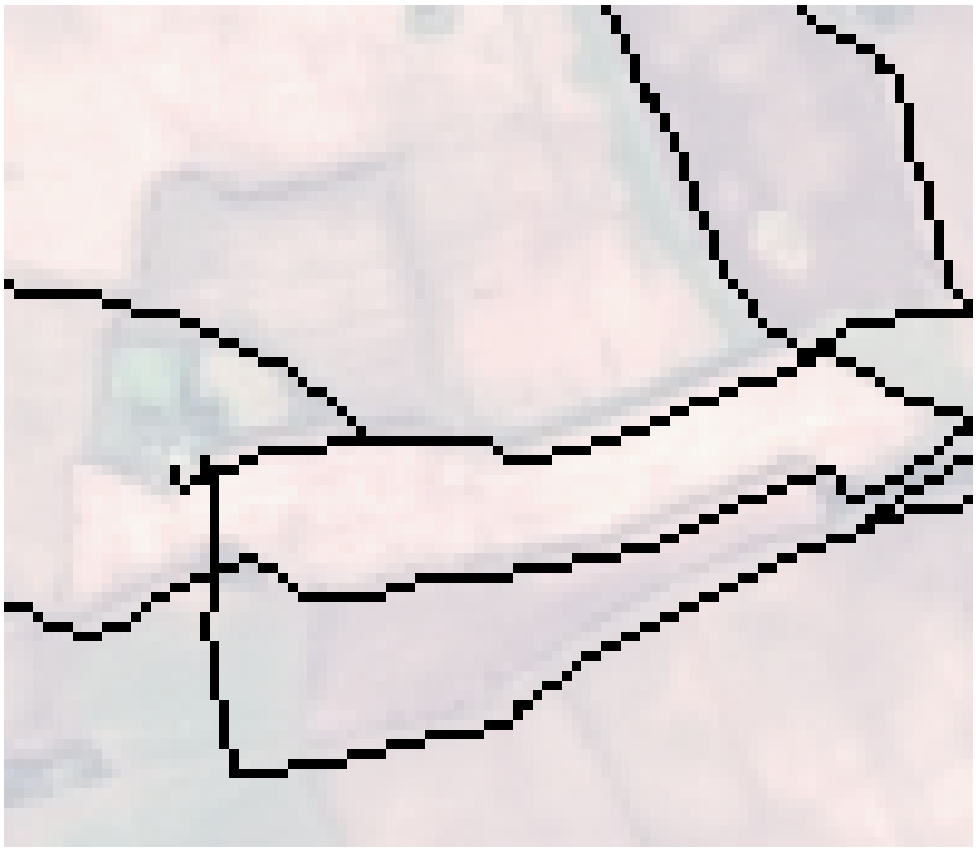
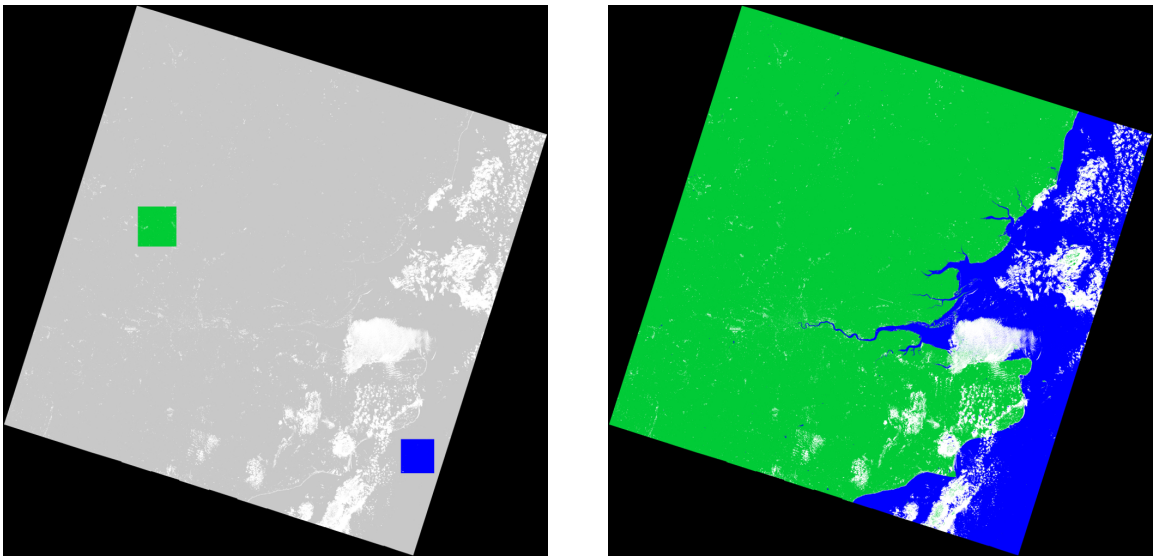


Figure 4.1: Misaligned and missing roads



(a) Manually drawn training map

(b) Resulting classification

Figure 4.2: Example of 'land-water' scenario, with cloud mask in white

4.3 Classification

4.3.1 Training the model

Pan sharpening

A multispectral image contains a higher spectral resolution than a panchromatic image, while a panchromatic image has a higher spatial resolution than a multispectral image. A pan sharpened image represents a combination between the multispectral and panchromatic images which combines the best of both image types, namely high spectral resolution *and* high spatial resolution [17]. An example of pan sharpening can be seen in Figure 4.3. Note that since the panchromatic band in the case of Landsat imagery is twice the resolution of the other bands, the pan sharpened image contains four times as many pixels as the non-pan sharpened image.

The Python package `landsat-util`[4] developed by *Developmentseed* specializes in downloading and processing Landsat images [3]. Their tools offer a convenient way to implement pan sharpening in any project and has helped a great deal in improving the accuracy of this method. Their tools have also been used to color-correct the pan sharpened images for previewing, as Landsat images come in grayscale only.

There are multiple ways to accomplish pan sharpening in the context of satellite imagery [1] - we did not further investigate different pan sharpening methods but assume that if a consistent method is used for all bands, resulting classifications will be similar. No methods currently exist to improve the resolution of the panchromatic image, so any pan sharpening method pulls up the resolution of the multispectral image to the resolution of the panchromatic band. The resolution of the resulting image can not be improved further with any of the aforementioned methods.

We apply pan sharpening to every band[5] using the panchromatic band as input. This results in 10 bands[6] with a resolution of 15 meters per pixel up from the 30 meters originally supplied by Landsat. From here on, we only consider pan sharpened images unless stated otherwise. In all of the following experiments pan sharpened images have been used. Note that a panchromatic band is supplied for Landsat 7 and Landsat 8 only. The pan sharpening step can be skipped for previous releases as it is not essential to our approach.

Random forest generation

A random forest is a collection of decision trees. In our model, the forest consists of 300 randomly generated trees. In Figure 4.4 we show that increasing the number of trees improves the accuracy of a model. Increasing the number of trees also lengthens the total runtime of our approach as shown in Table 4.1. The number of trees used for our model has been chosen arbitrarily.

A tree of a random forest is generated on a sample size of the original data set. This sample size is generated by drawing samples (with replacement) from the training set at random until the number of samples in the tree equals the total number of training samples. This process is

[4]Documentation at `https://github.com/developmentseed/landsat-util`

[5]Except the panchromatic band itself.

[6]The panchromatic band is discarded from the set of bands since its spectral resolution is contained in other bands and does not provide any additional information.

(a) Before pan sharpening **(b)** After pan sharpening

Figure 4.3: Example of pan sharpening

Number of trees	Training (minutes)	Classification (minutes)	Total (minutes)
300 trees	17	104	121
150 trees	12	49	61
10 trees	4	11	15
2 trees	2	9	11

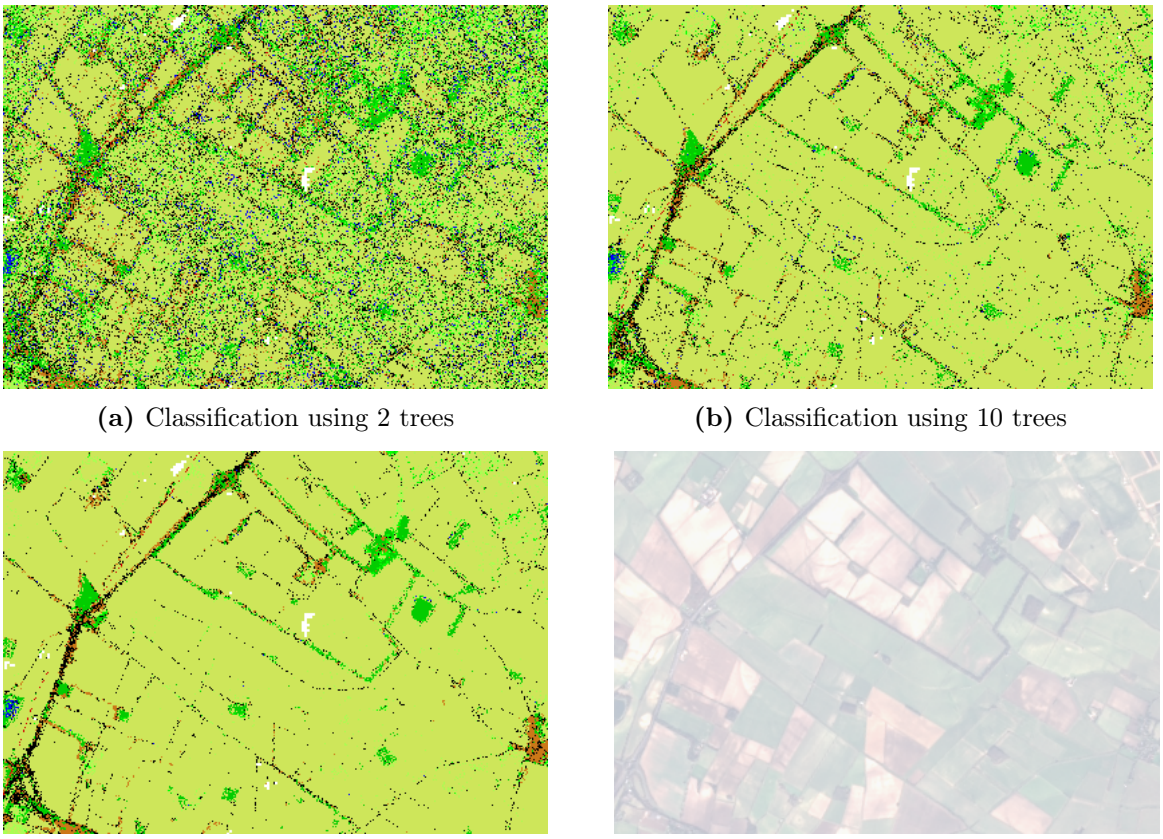
Table 4.1: Classification duration for varying number of trees

repeated for every tree in the forest. The way a single tree is generated for a random forest is the same as the generation of a regular decision tree[7]. These two properties of random forests imply a single decision tree will behave differently than a random forest with a single tree since the random forest has been trained on a slightly different set of samples. This also means that when choosing too few trees for a forest, any rare classes may not have been selected by the random selection algorithm, and cause the corresponding label to be excluded from classification.

Decision trees are generated by selecting a feature and a value for that feature that best split the training set in half. For both halves, this algorithm is repeated until every leaf in the tree contains only one sample. The corresponding parameters for the `RandomForestClassifier` class are as follows: `max_depth` controls the amount of layers of the tree. We are looking for a tree with only single samples in the leaves, resulting from setting this parameter to $None$. `min_samples_split` and `min_samples_leaf` are kept at the default value, being 2 and 1 respectively. Again, this is to make sure any leaf containing more than a single sample are split once more. The `max_features` parameter controls how many features are considered for every split. The default value for this is *auto*, which results in `max_features` $= \sqrt{n\_features}$[8]. For the 10 bands we are left with after pan sharpening, this means `max_features` $= 3$.

[7] Using Python package **sklearn**.

[8] Documentation at `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`



(a) Classification using 2 trees

(b) Classification using 10 trees

(c) Classification using 300 trees

(d) Actual scene

Figure 4.4: Classification using varying number of trees

Feature ranking

Some features may contribute to the outcome of a classification more than others do. Feature ranking shows which features are being used for splits more. The random forest shows which features are most important by counting the number of splits a feature is used for. This information follows after a model has been trained and the nodes of the tree can be traversed. Whenever a node is split using a certain feature, the importance of this feature is increased since it contributes to the outcome of the classification.

In the case of the 'all'-scenario (Figure 4.5), we see that features 10 and 11 are not used, i.e. are not informative. Features 10 and 11 correspond to bands 10 and 11 of the Landsat product, which are the two thermal bands. This implies the thermal bands do not contain useful information for classifying any of the classes in the scenario. This might be caused by the resolution being resampled from 100 meters which is, when compared to the resolution of the other bands, just not detailed enough for some of the classes with finer features.

After experimenting with a variety of scenarios, we found that bands 10 and 11 are not informative in any of the proposed scenarios. We did however continue any further experiments with band 10 and 11 as some method of classification might still make use of the bands. See Section 4.4 for a comparison of random forest classification to other approaches which might utilise the thermal bands.

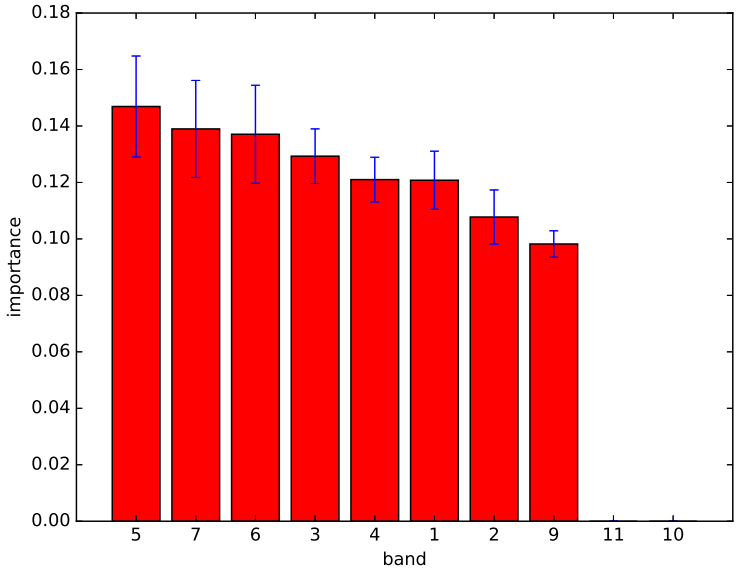


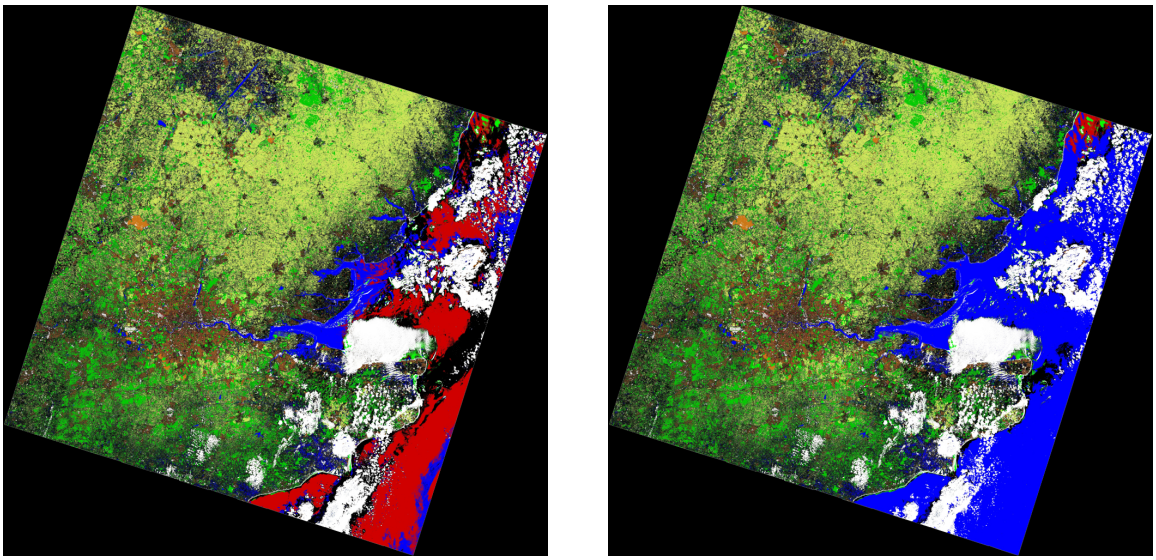
Figure 4.5: Feature ranking for 'all' scenario classification

Incorrect labels

We found an instance of incorrectly labeled pixels where our training data contained underwater railway tracks. In our case the channel tunnel railway mislabeled 'sea' pixels as 'rails' pixels. This would not have been a problem, if there would have been sufficient 'sea' pixels in our training data. However, the OpenStreetMap dataset did not include any of the North Sea pixels. This resulted in most of the 'sea' pixels being classified as 'rails' resulting in the classification in Figure 4.6a, with railways colored in red. This can be fixed by augmenting the original training data with a chunk of 'sea' pixels that significantly lowers the probability of sea being classified as railway tracks (or removing the rail pixels from the training samples). This small change in training data now results in Figure 4.6b which does not have this problem. This particular problem is discussed in more detail in Section 4.3.3 where we describe how the accuracy of OpenStreetMap (at least for this application) can not be guaranteed. This problem resulted not from pixels being labelled incorrectly in OpenStreetMap, but from our parser not removing railway tagged as underground from the data. This further reinforces the idea that results of our method will improve after careful manual inspection of the training data. We assume this problem is still present in our implementation to a lesser extent for overhangs, bridges, tunnels, and underground waterways.

4.3.2 Applying the model

Since Landsat satellite images do not cover the entirety of an image, filler pixels are included. Those pixels should not be used by the classifier to either train or classify since they do not represent actual terrain data. Those pixels can be discarded from the pixels fed to the classifier as discussed in Section 4.3.1.



(a) figure
Railway overclassification

(b) figure
No railway overclassification

Figure 4.6: Classification of London. No pan sharpening had been applied

Because of the size of the data, we have to classify the data in chunks since most computers would run out of ram. To do this, we simply split the 10 bands in chunks of x pixels where x is the number of pixels that fit in the computer's memory. The *predict* method for any scikit classifier works on chunks of any size. Not all classification methods suffer from memory limitations and related problems as we will see in Section 4.4.

After classification, we have an array of classified pixels. Together with the filler pixels from the QA band, a complete image can be generated. The resulting image does not contain any unclassified pixels. The fill mask may be substituted in the resulting image.

Cloud detection

Now that we have seen how we can manually label training data, we will continue implementing the cloud detection scenario. Labeling clouds is done by selecting several clouds on a satellite image. This resulting image will serve as input for the classifier. Figure 4.7 is an example of several labeled clouds. The polygons are filled with the color specified in the scenario definition file. After generating a traning image, the process of classifying clouds is exactly the same as for any other scenario. In Figure 4.8 we compare our results to the built-in cloud detection provided by Landsat. From this[9] we conclude that without manually labeling more training examples, cloud detection using our model is not as accurate as other methods [21]. Since we have labeled only the cloud centers, our method falls short on cloud edge detection. Manually tracing cloud edges is tedious, but would increase the performance of our method slightly. We did not further investigate improving cloud detection accuracy. Any classifications onwards use the built-in cloud detection.

[9]Comparing the results with the satellite images counting any discrepancies.

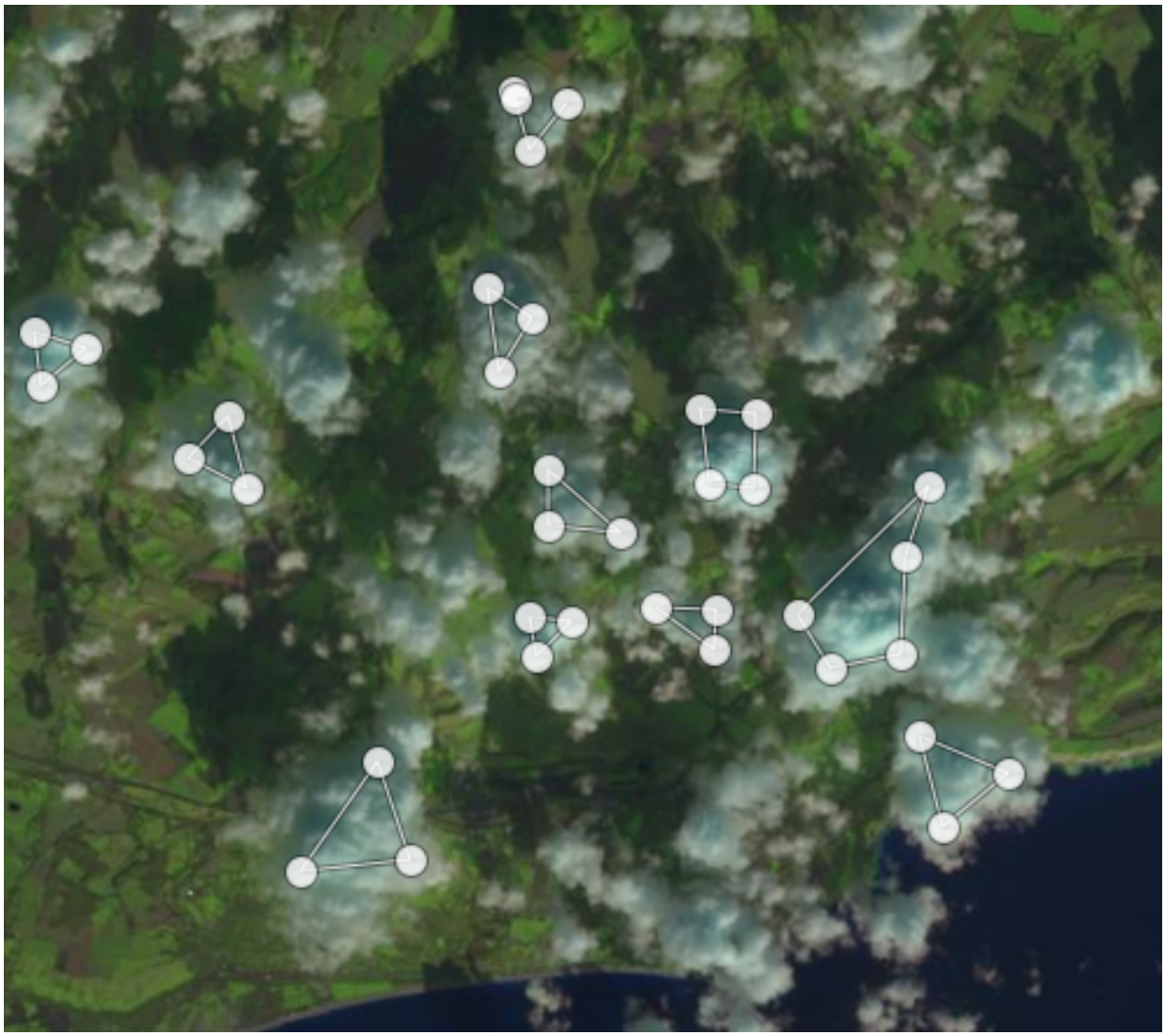


Figure 4.7: Labeling clouds

Re-using models

Dependent on normalisation of the different Landsat bands, we can re-use a trained model for multiple images. If we assume the model has been trained with sufficient coverage for each class, this model will work for any Landsat imagery. This is especially useful for areas where OpenStreetMap coverage is lacking, such as in third-world countries, or sparsely populated areas. It is preferred to use a model trained on an image taken of an area with comparable terrain conditions to the image to be classified[10]. It would not serve much purpose to classify an area with desert-like terrain with a model trained on a tundra-like area since the conditions differ too much. In Section 6.2.1 we will see how we can train our model using multiple Landsat images as input making it viable to generate a model that will perform well given any terrain type.

[10]The date of capture should also be similar when comparing two Landsat scenes, especially since the range of the thermal sensor values will change with the seasons.

(a) Result of the cloud detection scenario **(b)** Landsat's built-in cloud detection

Figure 4.8: Comparison of cloud detections

Due to improvements made to Landsat satellites over the years, re-using the model for different Landsat releases is not an option unless data that is not within the wavelength ranges of both satellites is discarded. For the transition between Landsat 7 and Landsat 8 two new spectral bands have been added, and some bands have had their wavelength range shortened (see Table A.1 for Landsat 7 band designations). Decreasing the wavelength range may result in different values for the same pixel across different Landsat releases. Our approach does not produce usable results out of the box when used across multiple Landsat releases. We expect that a translation of sensor values between releases can be developed to make this approach viable.

Unfortunately, a component of the Landsat 7 satellite has failed in 2003 rendering the subsequent data sets unusable for this project [12]. The Scan Line Corrector (SLC) is responsible for compensating the motion of the satellite so that the scan lines are properly aligned. After SLC failure approximately 22% of the data in any scene is missing. In Figure 4.9 we see that the image is crossed by lines containing no data. This data can not be recovered with post-processing which makes the entirety of the Landsat 7 imagery from 2003 onwards unusable for this application.

Quality assessment band

Another drawback of using imagery of older Landsat satellites is the absence of a quality assessment band. The Landsat acquisition plan of Landsat 7 has not changed for Landsat 8 meaning imagery is collected based on the same coordinates which implies we can use the quality assessment band from Landsat 8 to construct the filler mask for the former satellite. However, no cloud mask is present. We can substitute a mask generated by running a cloud detection scenario trained on Landsat 8 data. If this approach is chosen, improvements to our cloud detection scenario should be made so that it performs at least comparably to the built-in cloud detection as discussed in Section 4.3.2.

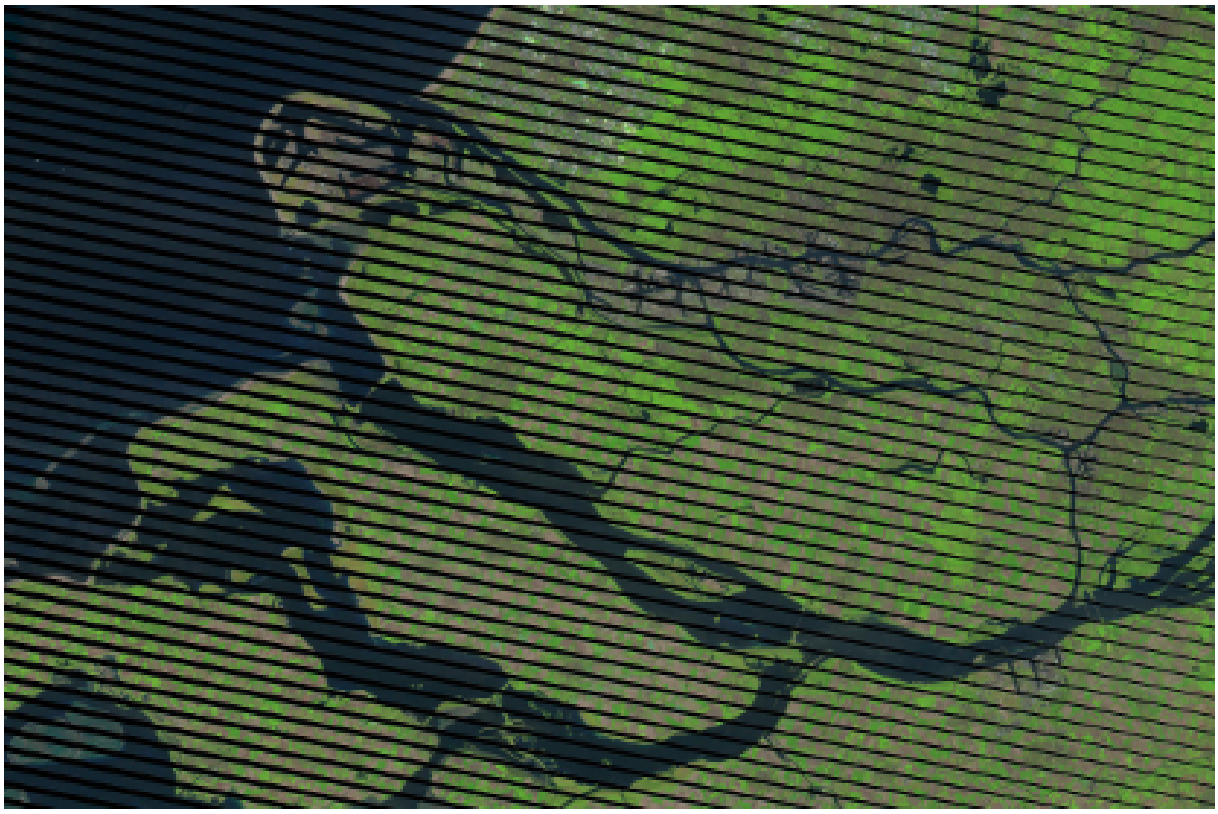


Figure 4.9: Landsat 7 SLC failure

This also poses the question on how to gather training data. Started in 2004, OpenStreetMap coverage for the early years of the project is severely lacking compared to the present day coverage. Historical dumps are available and can be used for older Landsat imagery. Training data can also be gathered from specialised sources for training a model on a specific part of the globe or for a specific purpose, but the absence of a worldwide, accurate set of training data makes for difficult training for older Landsat releases.

4.3.3 Performance

We can use the same OpenStreetMap data set that has been used to train the model, to test it. For this, the built-in method `score` for any sklearn classifier is used. For a scenario classifying land, water, and buildings[11] (Figure 4.10), the accuracy is 84%. This accuracy drops to 61% for a scenario defining six different features[12].

Confusion matrices

A confusion matrix is used to display which classes are most often misclassified, and what classes they are mistaken with. It is used to identify problematic classes that lower the performance of

[11] Buildings and roads grouped together.
[12] Water, roads, buildings, grass, forest, farmland

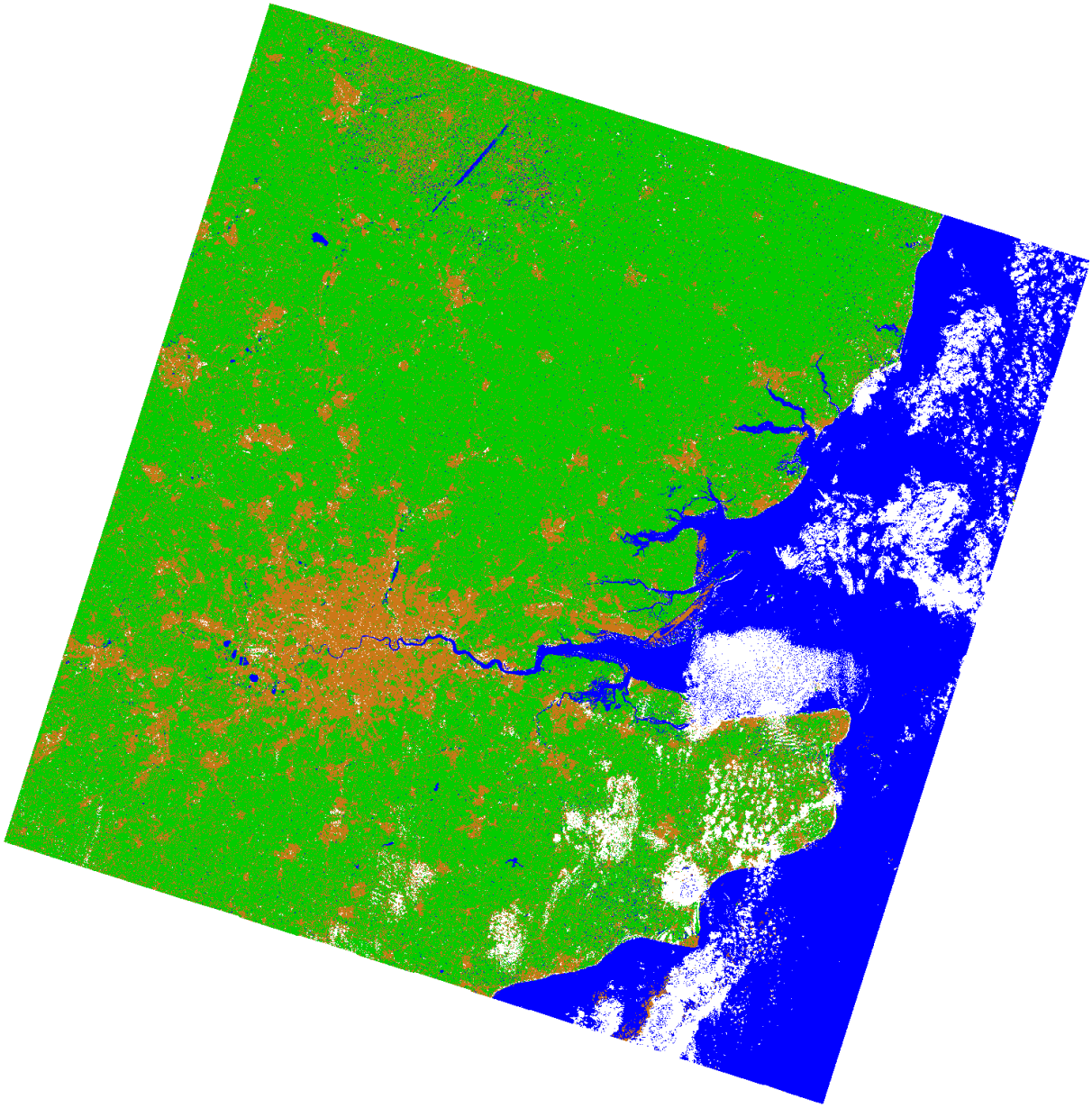


Figure 4.10: Results of classification for scenario defining land, water, and buildings

a classifier. For a scenario defining eight different features[13] the confusion matrix is plotted in Figure 4.11. From this matrix we can derive that:

- railways are almost never classified at all. This might be due to the same problem that road classification suffers, namely the low resolution of the imagery.
- the classifier easily mistakes roads and railways for buildings. This can be attributed to roads being close to, or flanked by, buildings, especially in cities. In the case of railways, underground railways also contribute to rails being classified as buildings.

[13]Features: water, grass, forests, parks, buildings, roads, railways, farmland.

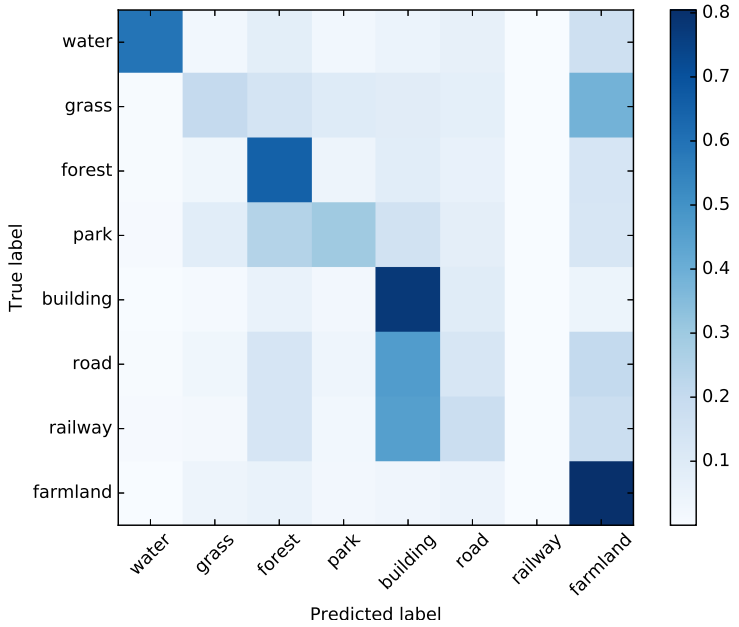


Figure 4.11: Confusion matrix for 'all' scenario

- due to the diversity of farmland, grass is often mistaked for it. Farmland includes a wide variety of crops, some of which show comparable thermal signatures to grass. Meadows are also often labeled under the more generic *farmland*-denominator in OpenStreetMap. Farmland is the class that is most often classified correctly in this scenario, being the darkest square in the matrix.

After looking at this confusion matrix, a decision was made to remove railways altogether. They suffer even worse from the resolution problem as railways are often more narrow than highways[14]. The grass and park classes were combined into one class. Resulting from those two changes is the confusion matrix in Figure 4.12.

OpenStreetMap

The performance of our method is highly dependent on the accuracy of the OpenStreetMap training data. As shown in [6], on average 88% of all highways defined in OpenStreetMap fall within 5m of the actual position of the road. This figure drops to 77% for smaller roads and motorways. The lower this figure, the more often the classifier will mistake the actual classes for roads. This might be caused by parts of the OpenStreetMap dataset being drawn from aerial imagery.

Problematic classes

Some classes are especially hard to classify. We will list some of the classes we found problematic here. Some of the problems might be solved by manually labeling these classes.

[14] Especially single-car tracks in forested areas

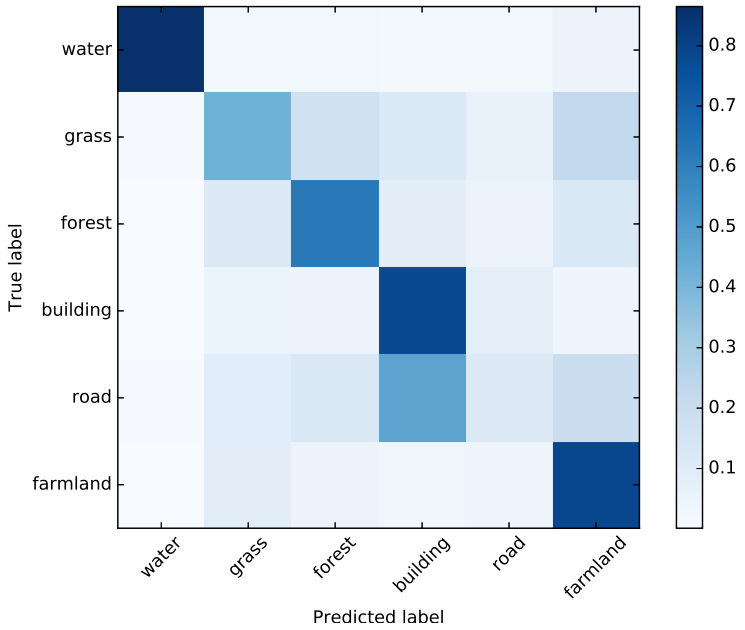


Figure 4.12: Confusion matrix for 'all' scenario with railways and parks removed

Snow and ice While the QA band includes a snow and ice mask, accuracy does not exceed 80% like the cloud mask. Often areas where terrain features are obscured by snow or ice, this cover is not present year-round. This makes classifying snow problematic since no good training labels can be found. Not all Landsat imagery can be used when images taken at periods where no snow is present have to be discarded. This greatly lowers the amount of training data we have available. We also assume snow detection suffers from the same problems as cloud detection, namely bright features are easily mistaken for it. Manually labeling instances of snow might be the best solution in this case. We did not classify any imagery with snow present.

Mountains In areas containing mountain ranges, shadows in crevices pose another problem. We found that shadows are most likely to be classified as roads. Moreover, bare stone is not labeled that often in OpenStreetMap, making this an especially tough class.

Cloud shadows As discussed in Section 4.1 shadows of clouds darken the tones of underlying pixels. Some cloud detection methods for Landsat include shadow detection algorithms [21], [2]. Our cloud detection scenario does not include such algorithms. The Landsat program is currently working on including a cloud shadow indication in the quality assessment band. It is indicated to be included with any Landsat 8 download from summer 2016 onwards [10]. We did not further explore the implementation of any of the aforementioned shadow detection algorithms to improve classification.

4.4 Comparison to other classification methods

We compare several common classification techniques to random forests.

4.4.1 K-nearest neighbor

The k-nearest neighbor (knn) implementation of scikit-learn[15] can be run using the same training maps as we have used for the random forest classifier. In this example we have used $k = 10$, meaning 10 neighbors are considered. The use of a knn classifier is not constrained by the amount of ram a machine has the way a random forest classifier is. The model can be fit using all available training data even for pan sharpened images. For certain use cases this approach is preferred as any outliers correctly marked in the training data are always considered whereas while constructing random forests some might be skipped due to the random nature of the selection algorithm. We have found knn classification for this application to be extremely slow due to the size of the data sets. Lowering the number of neighbors considered increases the runtime somewhat at the expense of accuracy.

4.4.2 Decision tree

Decision trees are the building blocks of random forests. Hence, training and classification times for a single decision tree are lower compared to random forests with multiple trees. For this comparison, the parameters equal the parameters of the random forest. What we found is that a decision tree overclassifies roads. See Figure 4.13 for a visual comparison.

4.4.3 Visual comparison

In Figure 4.13 we compare the results of the aforementioned classification methods to the random forest classifier. Both the decision tree classifier and the k-nearest neighbor classifier overclassify roads, while the random forest performed better.

4.4.4 Runtime comparison

In Table 4.2 we have collected runtimes for our method using various classification algorithms. These algorithms were run on the same Landsat imagery containing 264 million pixels using the same training data on the same machine. Since we are using the same training data for every run, the training image generation times are ignored[16].

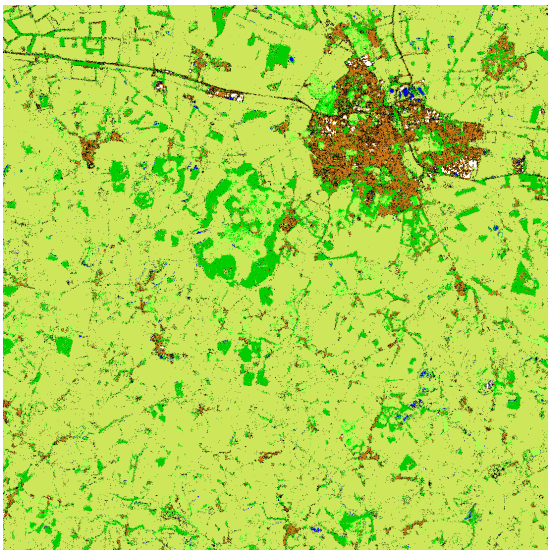
4.5 Statistics

Deriving statistics from classified images is a matter of counting pixels and their assigned labels. However, as we have seen with a resolution of 15 by 15 meters, we cannot make accurate statements about the surface area of every label. This goes especially for features where instances of the feature do not cover entire pixels. A pixel classified as road does not necessarily mean there is 225m2 of asphalt under it[17], while for a pixel classified as sea this is much more likely. Hence, we can only make an estimation of the amount of surface covered by a certain class.

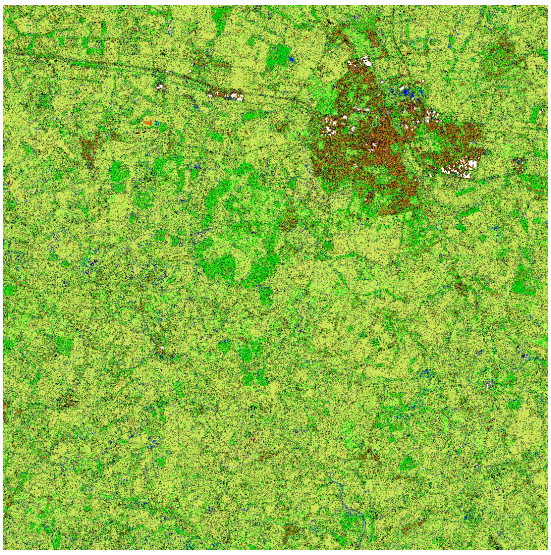
[15] Documentation at `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

[16] OpenStreetMap parsing and training map generation times are negligible at <1 minute.

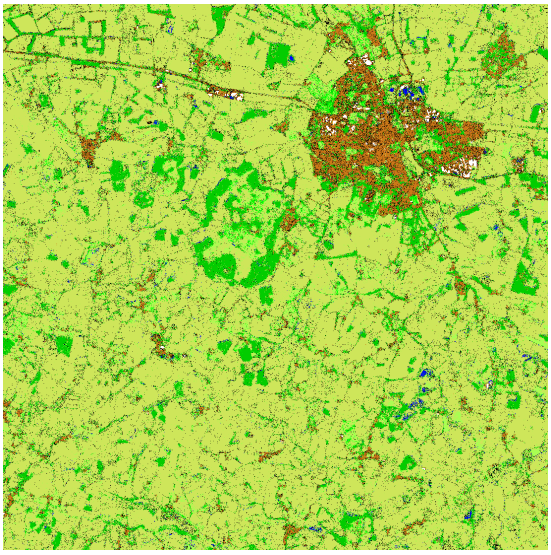
[17] Especially for dirt roads or bike paths.



(a) Random forest (150 trees)



(b) Decision tree



(c) K-nearest neighbor ($k = 10$)

(d) Actual scene

Figure 4.13: Classification results using different methods

Algorithm	Training (minutes)	Classifying (minutes)	Total (minutes)
Random Forest (300 trees)	17	104	121
Random Forest (150 trees)	12	49	61
Random Forest (10 trees	4	11	15
Random Forest (2 trees)	2	9	11
Decision tree	6	6	12
K-Nearest Neighbor (k=10)	8	361	369

Table 4.2: Comparison of various classification algorithms

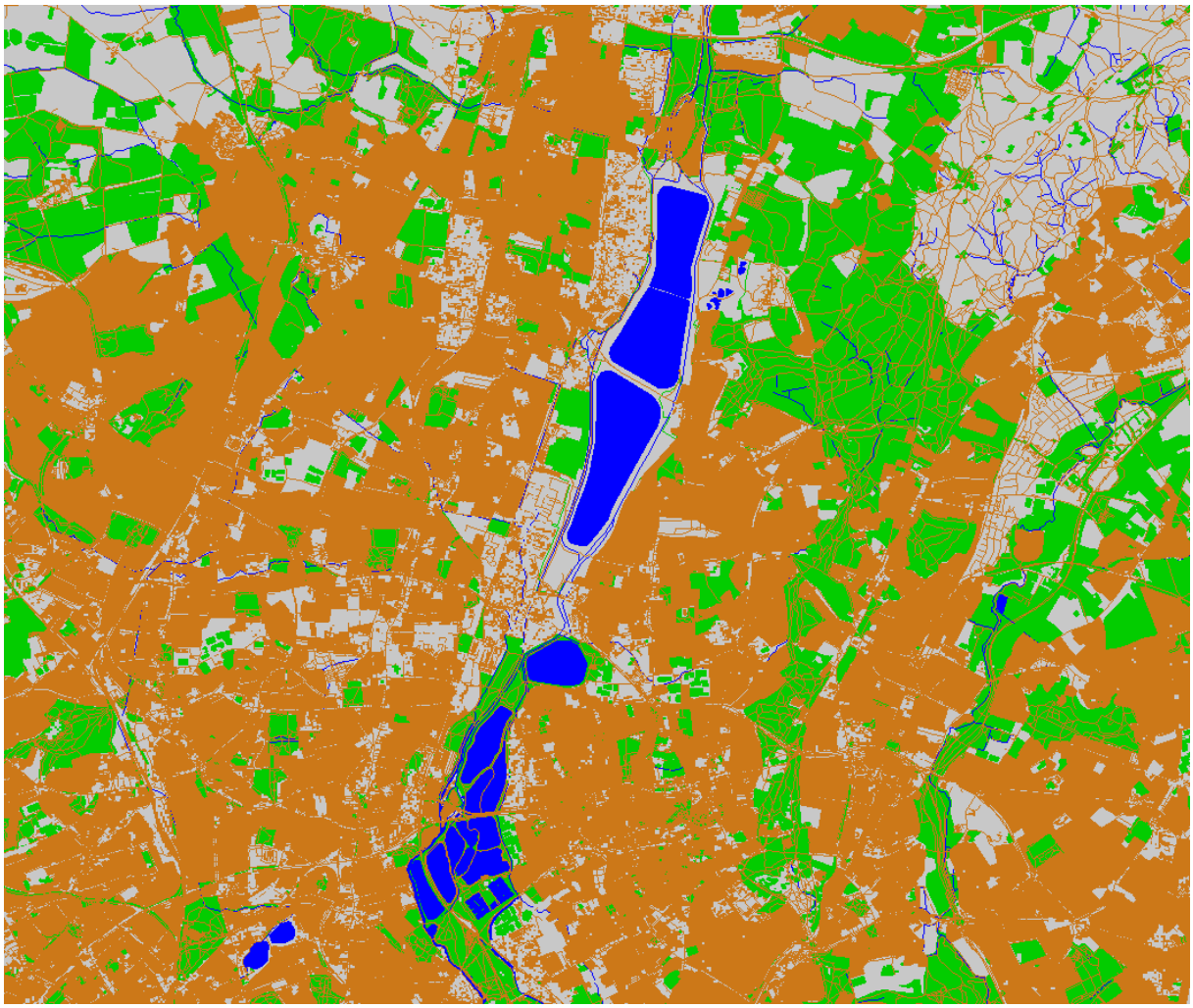


Figure 4.14: Training map for structure classification

This information combined with the longevity of the Landsat program means we can monitor the growth of cities in metropolitan areas. We propose a scenario that separates structures from nature for this very purpose. The structures class includes a variety of man-made objects like roads and buildings. The OpenStreetMap dataset can be used to plot the training map for this scenario. A subset of the training map can be seen in Figure 4.14. This subset shows almost complete OpenStreetMap coverage. Coverage of training data over the entire image was about 19%, leaving 81% of the image to be classified.

Because of the short period the Landsat 8 satellite has been online, we were not able to extract any meaningful statistics from this approach. However, as we have seen in Figure 4.3.2 this will become viable when the runtime of the Landsat program increases. Figure 4.10 shows the resulting classification for this scenario.

As with the training maps, we can extract statistics in two ways. First, pixel counts for the entirety of the classified image can be viewed (See Table A.2 for statistics on the entirety of Figure A.1). The second way makes use of the OpenStreetMap data set. OpenStreetMap dumps not only contain physical features, but also administrative boundaries of cities, states,

Class	Pixel count	Percentage
Water	1120	1%
Grass	24675	13%
Forest	8517	5%
Building	87397	48%
Road	27276	15%
Farmland	30327	16%
Clouds	3800	2%

Table 4.3: Pixel counts for Cambridge

and countries. We plot the administrative boundaries to a separate image file to be used as input for the statistics script. Figure 4.15 is an example of this method for the administrative boundary of Cambridge overlaid on a satellite image. Statistics for every pixel inside this boundary are now calculated. Again, using this approach ensures we can also manually draw boundaries or areas. The results of this count are listed in Table 4.3. We see that most of the area is covered by buildings which is expected for cities.

4.5.1 Limiting factors

The accuracy of statistics is dependent on several factors that influence the pixel count of any given scene.

Clouds We include the number of cloud pixels in the final pixel count. The usefulness of the statistics decrease as the number of cloud pixels increase as it is impossible to accurately predict the actual class of the cloud pixels.

Classifier performance With decreasing performance, accuracy of pixel counts also decrease. Confusion matrices can help to identify problematic classes. If a confusion matrix shows which classes are mistaken for which other classes, their counts can be modified accordingly.

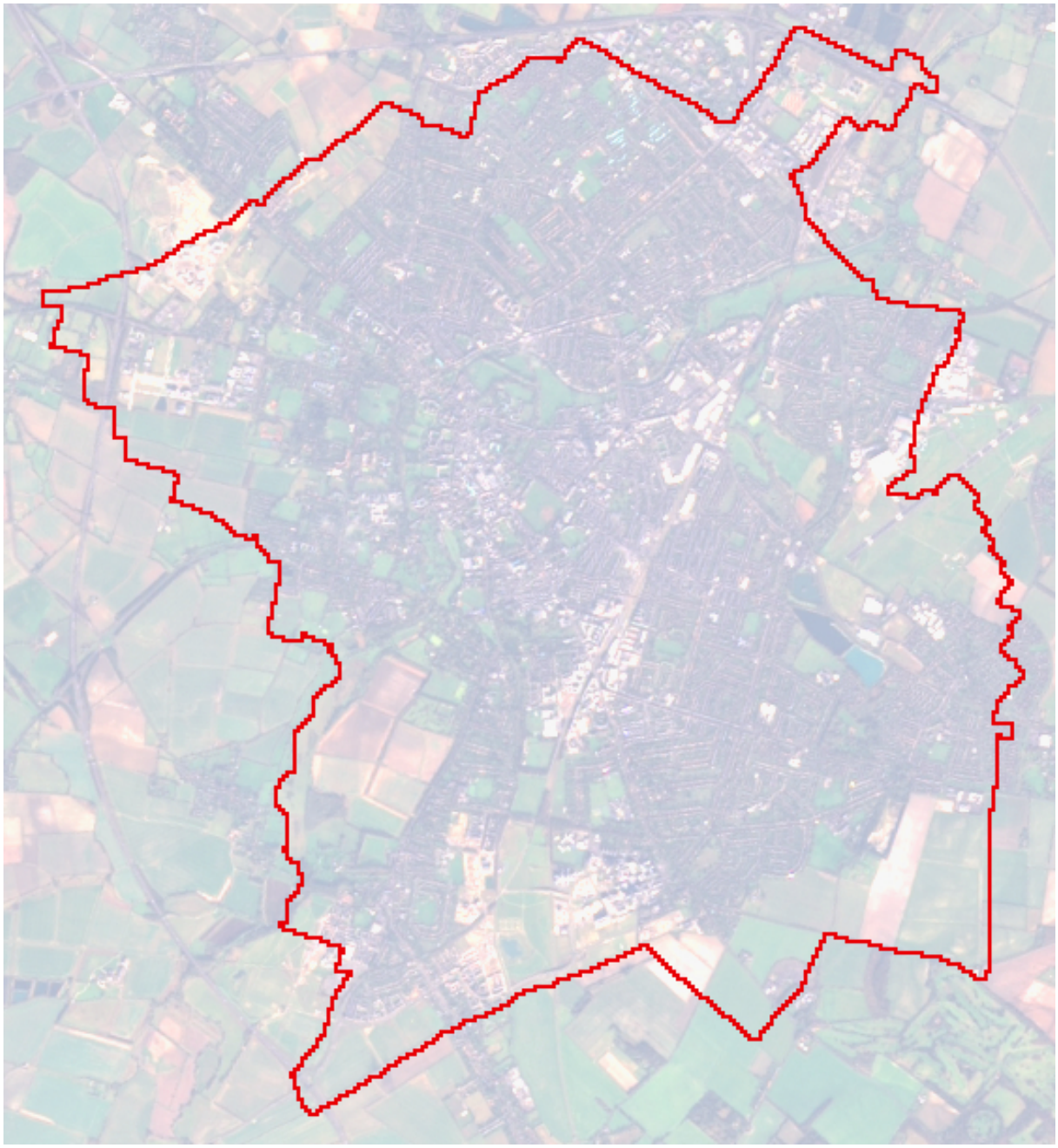


Figure 4.15: Administrative city boundary of Cambridge

5 | Related Work

In the field of satellite image classification, many papers have been written. We will list a few methods used in the most interesting ones here.

5.1 Training samples generation

While some land cover mapping methods utilise OpenStreetMap for their training data [7] [18], no automated approach for mapping training data to satellite images is used. This improvement greatly reduces the time needed for creating training data, as drawing training maps from OpenStreetMap dumps is done in under a minute.

5.2 Different wavelengths

Some methods focus on the red, green and blue channels of the images only. Using these three bands, the number of features used for classification starts at three[1]. For older Landsat releases, this is not an implementation limitation but rather a limitation of data. Our method differs in that it uses all available bands, and in some cases might add more bands (i.e. an edge band) which should improve performance of the model. Methods like this using only three bands can be applied to images from any source meaning the spatial resolution of an image is not an issue anymore.

Methods for differentiating between multiple agricultural land uses have been developed. We did not implement such features as gathering training data would be too tedious and prone to change over the years due to crop rotation. For most of the farmland mapped on OpenStreetMap, no distinction between crop varieties or even farmland used for grazing is made. This reason was one of the factors for deciding whether or not to create a scenario for classifying crop types.

[1] The number of features can be increased as we have seen in Section 6.1.2.

6 Future work

In this section we will describe future work that we think can improve the accuracy of the method.

6.1 Classification improvement

There are several ways we can improve our method with. Those methods fall outside the scope of this thesis but we will discuss various ideas we think would result in better performance for our method. Some of the methods mentioned here have been implemented before as we have seen in Chapter 5.

6.1.1 Surrounding pixels

We can take into account surrounding pixels when training our model. For 8 surrounding pixels this would increase the number of features from the original 10 bands to 10 bands of the center pixel plus 8 times 10 bands for the surrounding pixels. Improved random forest algorithms may be required for this approach to be viable on any modern pc. We can assume this method will increase the accuracy of classes with large surface areas such as bodies of water since for any pixel surrounded by all water pixels, it is very likely this pixel is also water.

6.1.2 Edge detection

Edge detection is a method to pinpoint pixels where brightness changes sharply. One application of edge detection in our method is road detection. For a number of bands the brightness for roads and neighboring farm fields or grass lands changes sharply. This might also be the case for buildings and their surroundings. We can store the edges of certain bands in a separate band to use as training data for our classifier. When implementing edge detection, we can choose any number of bands to apply edge detection to, resulting in as many edge detected bands as original bands. Figure 6.1 is an example of the average edge over bands 2, 3, and 4.

6.1.3 Label noise

Wrongly labeled pixels in the training data result in misclassified pixels in the satellite image. We currently have no way of detecting OpenStreetMap label noise except manual verification since

Figure 6.1: Edge detection applied to Figure 4.3b

no other open datasets with comparable coverage exist. Because of the crowdsourcing nature of the project, users might not be entirely accurate when labeling terrain features. Figure 4.1 is an example of misaligned roads. On the other hand, riverbanks can shift over time, making OpenStreetMap misalign the riverbank with the actual position of the river. Such errors are likely not caused by human mistake. Some work has been done on improving the performance of models trained with noisy data [14]. Implementing those methods in our model would likely result in better performance. We might also improve the performance of our model by rejecting any OpenStreetMap data that has not been updated or reviewed recently. Currently, no data on time of entry or modification is supported by the OpenStreetMap data dumps.

6.2 Multi-image classification

6.2.1 Mosaicing images

Our approach works for single Landsat dumps only. Since we have some information on the location of where the image was taken, we can stitch together Landsat dumps to train and classify multiple images as if they were one. Different weather conditions should be taken into account whenever two images taken on different dates are used as the cloud cover and brightness may vary. This approach can be automated as the USGS provides an API [4][1] to programmatically search and download large quantities of Landsat imagery. OpenStreetMap also provides such an API [16]. This approach, combined with the administrative borders OpenStreetMap provides, can be used to obtain statistics for large countries in one pass.

[1]Login required for bulk downloads.

6.2.2 Merging images

For any image with a percentage of cloud covered pixels, we can find a Landsat image of the same scene taken on a different date with different cloud coverage. For two (or more) images with varying degrees of cloud cover, this means we can select pixels from either image to obtain an image where no cloud cover is obscuring parts of the scene. We can then use this image to train and classify with our model. We have to keep in mind the shadows cast by the clouds as they might have to be removed also. We also have to keep in mind the conditions the images are taken under as changing weather can influence sensor values. This approach can be especially useful for extracting statistics since cloud pixels cannot be classified and counted.

6.3 Improving statistics

Increasing resolution We can attempt to improve the statistics by increasing the resolution of the scenario training image independent of the corresponding Landsat imagery. This is especially useful for classifying features smaller than a certain threshold since features can seem to blend together when the resolution is not high enough. Take for example a highway, where overhanging trees might obscure part of the highway and thus label their leaf color as highway in the resulting model. When a higher resolution training image is used, we can choose to ignore the outer shell of pixels for wide roads eliminating this problem. This increased resolution of the training map comes at no significant computation cost since the number of pixels that a model is trained on does not increase. An algorithm to select suitable pixels from the training map has to be developed for this to work.

7 | Conclusions

We have presented a way to classify terrain data using Landsat 8 imagery and OpenStreetMap. Our approach shows flexibility in choosing which labels to classify, based on OpenStreetMap data. This method can be applied to any Landsat release, as long as suitable training maps can be created.

We have also seen that road detection proves difficult. This is partly due to the resolution of the Landsat imagery. Even after applying pan sharpening, which improves results somewhat, we found that road is often confused for buildings. Comparing Figure 4.6b to Figure A.1, it is obvious that roads (in black) are not classified as often in the pan sharpened result. No other parameters were changed.

Either of the two methods for generating training data can be selected, based on the application. Especially for cloud detection, we require the manual method as OpenStreetMap does not label clouds. With this method, we have shown that cloud detection is possible. While it did not directly improve over the built-in cloud detection with the training data we generated, with more detailed labeling comes better performance.

Bibliography

[1] Ilham Alimuddin, Josaphat Tetuko Sri Sumantyo, Hiroaki Kuze, et al. Assessment of pan-sharpening methods applied to image fusion of remotely sensed multi-band data. *International Journal of Applied Earth Observation and Geoinformation*, 18:165–175, 2012.

[2] Hyeungu Choi and Robert Bindschadler. Cloud detection in landsat imagery of ice sheets using shadow matching technique and automatic normalized difference snow index threshold value decision. *Remote Sensing of Environment*, 91(2):237–242, 2004.

[3] Satellite imagery power tools. `https://developmentseed.org/projects/landsat-util/`. Accessed: 10-07-2016.

[4] Earthexplorer. `http://wiki.openstreetmap.org/wiki/API_v0.6`. Accessed: 21-05-2016.

[5] Epsg geodetic parameter registry. `https://www.epsg-registry.org/`. Accessed: 12-05-2016.

[6] Mordechai Haklay. How good is volunteered geographical information? a comparative study of openstreetmap and ordnance survey datasets. *Environment and planning B: Planning and design*, 37(4):682–703, 2010.

[7] Brian A Johnson and Kotaro Iizuka. Integrating openstreetmap crowdsourced data and landsat time-series imagery for rapid land use/land cover (lulc) mapping: Case study of the laguna de bay area of the philippines. *Applied Geography*, 67:140–149, 2016.

[8] Landsat applications. `http://landsat.gsfc.nasa.gov/?p=3501`. Accessed: 16-07-2016.

[9] What are the band designations for the landsat satellites? `http://landsat.usgs.gov/band_designations_landsat_satellites.php`. Accessed: 08-03-2016.

[10] Landsat quality assessment band. `http://landsat.usgs.gov/collectionqualityband.php`. Accessed: 02-08-2016.

[11] Landsat quality assessment band. `http://landsat.usgs.gov/qualityband.php`. Accessed: 08-03-2016.

[12] Slc-off products: Background. `https://landsat.usgs.gov/products_slcoffbackground.php`. Accessed: 29-07-2016.

[13] What are the best spectral bands to use for my study? `http://landsat.usgs.gov/best_spectral_bands_to_use.php`. Accessed: 08-03-2016.

[14] Volodymyr Mnih and Geoffrey E Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 567–574, 2012.

[15] Doug R Oetter, Warren B Cohen, Mercedes Berterretche, Thomas K Maiersperger, and Robert E Kennedy. Land cover mapping in an agricultural setting using multiseasonal thematic mapper data. *Remote Sensing of Environment*, 76(2):139–155, 2001.

[16] Elements - openstreetmap wiki. `http://wiki.openstreetmap.org/wiki/Elements`. Accessed: 13-05-2016.

[17] Arya Krishnan P.S and Chithira Rakshmi G. Comparative study on pansharpening methods for satellite images. *International Research Journal of Engineering and Technology*, 2, 2015.

[18] Quick land cover estimates from satellite imagery and openstreetmap. `http://rexdouglass.com/quick-and-dirty-land-cover-estimates-from-landsat-satellite-imagery-and-openstreetmap/`. Accessed: 05-12-2015.

[19] Yuliya Tarabalka, Mathieu Fauvel, Jocelyn Chanussot, and Jón Atli Benediktsson. Svm-and mrf-based method for accurate classification of hyperspectral images. *IEEE Geoscience and Remote Sensing Letters*, 7(4):736–740, 2010.

[20] Billie Turner, David Skole, Steven Sanderson, Günther Fischer, Louise Fresco, and Rik Leemans. Land-use and land-cover change. In *International Geosphere-Biosphere Programme, Stockholm; Report, 35*, 1995.

[21] Zhe Zhu and Curtis E Woodcock. Object-based cloud and cloud shadow detection in landsat imagery. *Remote Sensing of Environment*, 118:83–94, 2012.

A | Appendix

A.1 Classification results

A.1.1 London

Figure A.1 is the classification result for the Landsat imagery with identifier `LC82010242016047LGN00` taken on the 16th of February, 2016.

A.1.2 New York

Figure A.2 is the classification result for the Landsat imagery with identifier `LC80130322016106LGN00` taken on the 29th of April, 2016.

A.2 Landsat miscellaneous

Table A.1 contains the Landsat 7 band designations. Comparing to Table 3.1 shows that our implementation does not work when the training and classifying data set are sourced from different Landsat releases.

Bands	Wavelength (μm)	Resolution (meters)
Band 1 - Blue	0.45 - 0.52	30
Band 2 - Green	0.52 - 0.60	30
Band 3 - Red	0.63 - 0.69	30
Band 4 - Near Infrared	0.77 - 0.90	30
Band 5 - SWIR 1	1.55 - 1.75	30
Band 6 - Thermal Infrared	10.40 - 12.50	30
Band 7 - SWIR 2	2.09 - 2.35	30
Band 8 - Panchromatic	0.52 - 0.90	15

Table A.1: Landsat 7 band designations

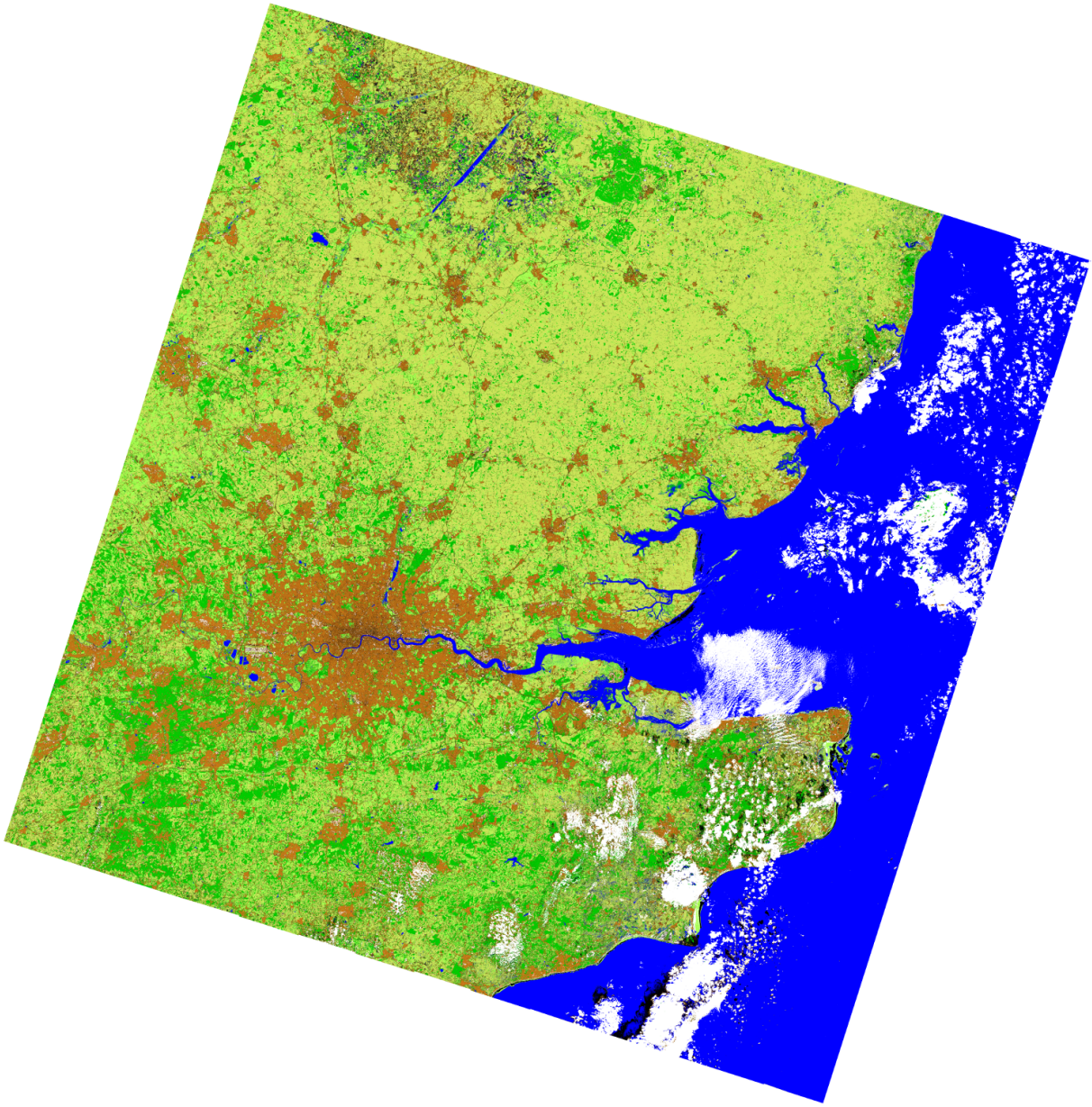


Figure A.1: Result of classification for London

Class	Pixel count	Percentage
Water	34616462	21%
Grass	19480650	12%
Forest	20880638	12%
Building	20128139	12%
Road	10648960	6%
Farmland	50929270	31%
Clouds	10199353	6%

Table A.2: Full statistics for image Figure A.1

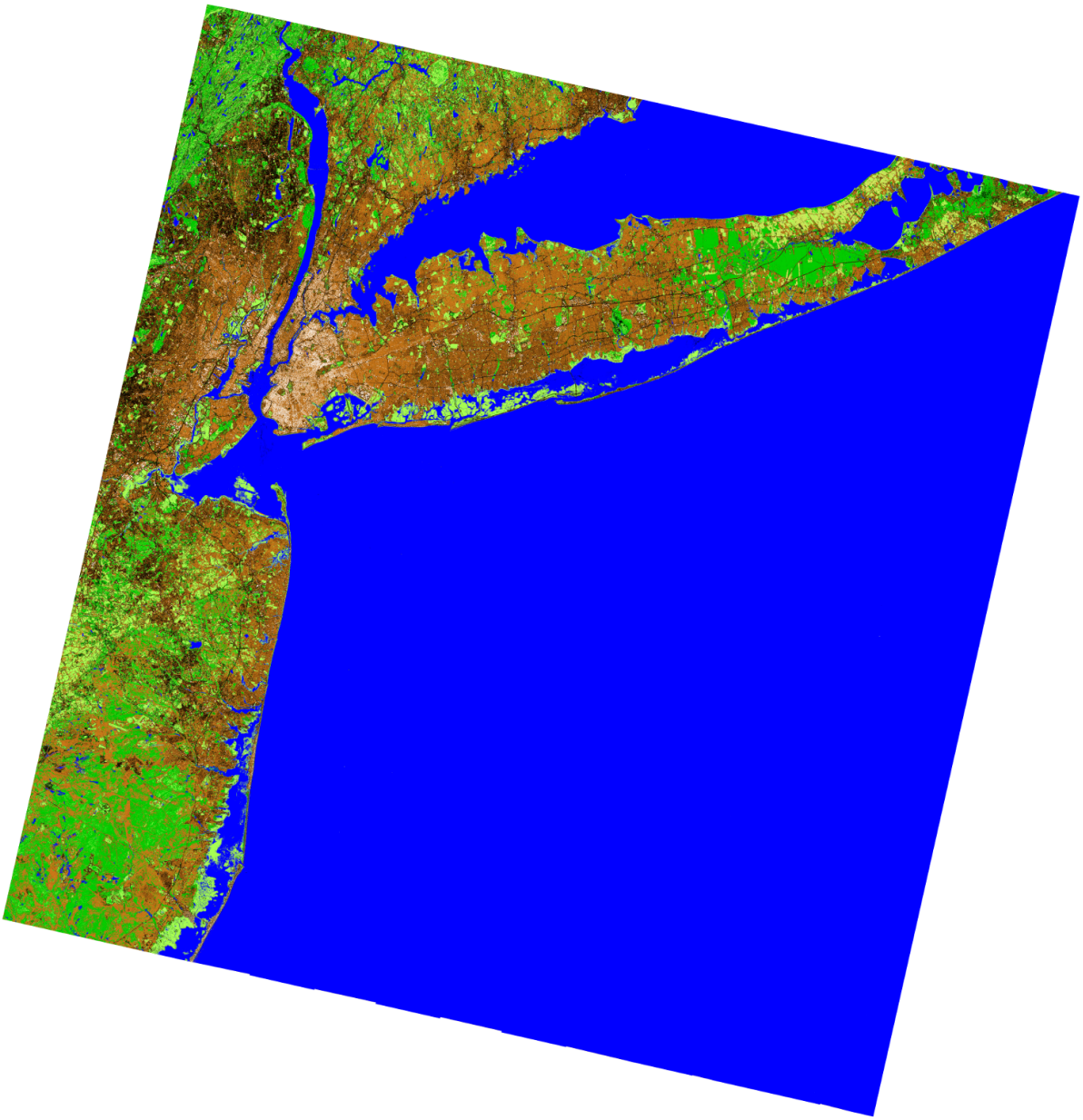


Figure A.2: Result of classification for New York

A.3 Data sources

A.3.1 Landsat

All Landsat imagery has been downloaded using the online portal *EarthExplorer* hosted by the USGS accessible at `http://earthexplorer.usgs.gov/`. Note that an account is required for full access to every available Landsat download. For browsing the website an account is not needed.

A.3.2 OpenStreetMap

While OpenStreetMap hosts downloads for their data set on the official OpenStreetMap website, the data is not split into multiple files unless their API is used. For smaller subsets of the OpenStreetMap data set, Geofabrik hosts an excellent tool to browse and download regions. Their website is accessible at `http://download.geofabrik.de/`. No account is required.